2009

3D Hand gesture recognition using a ZCam and an SVM-SMO classifier

Lucas Bonansea
Iowa State University

3D Hand gesture recognition using a ZCam and an SVM-SMO classifier

by

Lucas Bonansea

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Co-majors: Computer Science; Human Computer Interaction

Program of Study Committee:
Vasant G. Honavar, Co-major Professor
Stephen Gilbert, Co-major Professor
Leslie L. Miller

Iowa State University

Ames, Iowa

2009

Table of Contents

List of Tables

List of Figures

Abstract

The increasing number of new and complex computer-based applications has generated a need for a more natural interface between human users and computer-based applications. This problem can be solved by using hand gestures, one of the most natural means of communication between human beings. The difficulty in deploying a computer vision-based gesture application in a non-controlled environment can be solved by using new hardware which can capture 3D information. However, researchers and others still need complete solutions to perform reliable gesture recognition in such an environment.

This paper presents a complete solution for the one-hand 3D gesture recognition problem, implements a solution, and proves its reliability. The solution is complete because it focuses both on the 3D gesture recognition and on understanding the scene being presented (so the user does not need to inform the system that he or she is about to initiate a new gesture). The selected approach models the gestures as a sequence of hand poses. This reduces the problem to one of recognizing the series of hand poses and building the gestures from this information. Additionally, the need to perform the gesture recognition in real time resulted in using a simple feature set that makes the required processing as streamlined as possible.

Finally, the hand gesture recognition system proposed here was successfully implemented in two applications, one developed by a completely independent team and one developed as part of this research. The latter effort resulted in a device driver that adds 3D gestures to an open-source, platform-independent multi-touch framework called Sparsh-UI.

1. Introduction

The rapid evolution of computer-based technology and the growing number of complex applications have increased the need for more natural means of interaction between human users and computer systems. Although keyboard and mouse have been successfully used as main interfaces in many applications, more complex applications require more natural means of interaction. Gestures are one of the most natural forms of human interaction, so they offer a good solution for those applications.

Using gestures as an interface with complex applications would allow users to interact with visually complex systems. Using a mouse and keyboard is difficult in these systems because they are designed for working in a 2D space, whereas generally visually demanding systems require users to interact in a 3D space. Anther interesting context in which gestures seem to fit better than keyboard and mouse is a public setting or conference when the user is required to play a more active role with respect to the environment.

There are two main groups of gesture interfaces with computer-based applications: those based on touch surfaces or stylus-based tablets, and those based on computer vision (cameras). Touch surface gestures are already part of people's everyday life—examples include touch phones like Apple's iPhone, the Samsung SGH-F480 or the HTC P347, and CNN's “Magic Wall” used during the 2008 U.S. presidential elections. Although great effort has been invested in computer vision-based gestures, they have not yet reached the popularity of touch-based gestures. Possibly this is because of the great difficulty that computer vision systems have with understanding the scene presented by the camera. The availability of new 3D hardware appears to be an important step in mitigating this problem.

1.1. Motivation

The goal of this work is to propose a complete solution for the one-hand 3D gesture recognition problem using a 3D camera called ZCam [12] and Support Vector Machines [4]. The solution claims to be complete because it focuses on recognizing a gesture and understanding the scene so users can start and stop gestures at any moment—other special postures or interactions such as a keyboard or mouse are not necessary. Additionally, this research has a secondary goal to consider the importance of being able to recognize gestures in real time (which requires processing to be as simple as possible).

Thus, the specific research question is:

Is it possible to create a complete gesture recognition system using the ZCam which recognizes previously-trained hand gestures in real time and does not require any other interaction from the user?

The selected approach to address these goals is to model the 3D gestures as a sequence of poses, reducing the problem to a hand pose recognition problem. However, hand pose recognition by itself is not enough to achieve the goal of developing a complete system. To understand the scenario at all times, hand recognition and tracking must also be considered. Finally, the hand pose recognition problem was refined into pose and rotation recognition where the pose of the hand is considered independent of the rotation angle.

1.2. Related Work

Computer-based applications are constantly evolving into more complex systems that require a more active interaction with the user. In this context, keyboard and mouse are no longer ideal interfaces; there is an increasing need for more natural interfaces. Gestures are one of the most natural means of communication between humans, so it is not surprising that

human computer interaction based on gestures has become an important theme of research in recent years.

Research in computer vision-based gestures had a strong breakthrough in the early to mid-1990s. It remains an active research area, and much work has been done over the last few years. The more advanced hardware now available provides an opportunity for many new advancements. To address the related work, it is better to divide the research into classes, although these classes are not completely separate. One approach is to consider how gestures are modeled for recognition. In this sense, there are two main groups of researchers: those who consider the gesture as a sequence and focus their efforts on comparing such sequences, and those who consider the gesture to be the sum or concatenation of several static poses and focus their effort on recognizing each static pose.

For the group which considers the gestures dynamically in sequences, the general approach is to consider different feature sets to train Hidden Markov Models (HMM) classifiers or similar algorithms. This approach is interesting because it deals with a series of movements in a way that is less susceptible to low image resolution or to losing some frames during the execution of the gestures. However, this approach is more susceptible to the execution of the gesture, so when new users who are not well trained don't perform the gesture perfectly, this type of classifiers has a hard time. Some relevant researchers using this approach are Pentland, Sclaroff, Starner, and Wei. [35], [36], [7], [11], [2].

Starner and Pentland [35] propose extending the use of HMM from speech and handwriting recognition to visual gesture recognition. By doing so, they were able to model American Sign Language (ASL) gestures without modeling hands and fingers. Starner, Weaver, and Pentland [36] extended their previous work to recognize ASL sentences,

tracking the user's unadorned hands using a desk-mounted camera and a wearable camera attached to the user's cap.

Chen et al., [11] use hand tracking and movement detection to identify candidate regions where a feature set based on spatial and temporal information is extracted and used to feed a HMM classifier for gestures recognition. Alon et al., [2] recognize that using techniques like skin color detection, movement analysis, and background subtraction can be very helpful for gesture recognition, but they are not reliable with more complex backgrounds so they propose to wrap the results of the skin color detection and movement analysis into time sequences and then do their gesture recognition by comparing such sequences. To do this, they propose an extension of the dynamic time wrapping algorithm called Dynamic Space-Time Wrapping (DSTW) algorithm.

Researchers who consider the gestures as a combination of static poses focus their work on different means of processing and recognizing hand poses in each frame. Then the gesture is built by combining the previously obtained information. In this group, the preferred classification mean is diverse, ranging from statistical analysis to image-based classification like eigenspaces to machine learning classifiers like neural networks or Support Vector Machines (SVM). One advantage of this approach is that it builds the gesture based on the known poses. Once the classifiers are trained in a given set of poses, a considerable number of gestures can be built. Actually, new gestures can be built without needing new training, as long as no new hand pose is required. Some prominent authors in this group are Huang, Strintzis, Stenger, Neumann, and Wu [39], [40], [3], [43], [5], [6], [20], [1], [23]. Moreover, it is worth noting that Tseng, Sun, and Jiang [37], [42], [18] use SVM as their preferred classification method.

Wu and Huang [39], [40] recognize the intrinsic difficulties of modeling the human hand and its articulations. In [40], they propose an appearance-based learning approach to handle large variations of linear points. To alleviate the learning process, they use a combination of supervised and unsupervised learning paradigms and a large number of unlabeled training samples. In [39], they propose a two-step iterative, model-based algorithm to capture articulated human hand and motion.

Shan et al., [6] integrate two successful visual tracking approaches such as particle filtering and mean shift to improve their hand tracking process. Patwardhan and Roy [20] propose an eigenspace framework which models hand gestures based on both hand shape and motion tracking. Al-Rajab et al., [1] and Gu and Su [23] use Zernike moments for gesture recognition.

Chen and Tseng [37] use a combination of three SVM classifiers to recognize multiple-angle hand postures in finger guessing games. Liu et al., [42] propose an algorithm based on Hu moments and SVM to recognize hand postures and evaluate whether or not the hand can meet the requirements of a driver's license test. Finally, Ye et al., [18] combine the greater classification power of SVM (when dealing with good generalization properties and limited samples) with HMM (which are good for dealing with sequences) to recognize Chinese sign language.

Another relevant method of classifying the current work done on Computer Vision gestures is to study the input data used to analyze the gestures. Again, there are two main groups: those that work with 2D streams and those that have available 3D information about the scene.

Working in 2D has an important advantage in that it uses the least expensive hardware that

is available. However, the problem of analyzing the scene is more difficult. Several techniques are used to address the problem of understanding the scene such as skin color detection, movement analysis, and background removal. Although many of these techniques have been refined throughout the years, they are still not completely reliable. Background objects have colors which are easily confused with skin color, and background noise can disturb both movement analysis and background removal. Nonetheless, new algorithms to mitigate these problems are being developed and published. Some authors who have done interesting work in this group are Cheng, Lu, Collobert, and Xu. [38], [25], [21], [34], [9].

Starting with an specific gesture, Fang et al., [38] use motion and color cues to perform hand detection and tracking. Fujimura and Xu [21] address the problem of recognizing those sign language signs which include hand overlapping by converting the input blob into a graph that represents the finger and the palm of the user's hand and processing the new graph by either subdivision or integer programing.

Completed work based on 3D input has increased in recent years, as the required hardware is now more accessible [22], [12]. There are many advantages to working with 3D input data because many of the background problems that are difficult to solve in a 2D environment are easily avoided with the new information. In general, the main concern when working with 3D data is how to use the depth information to separate the relevant information of the scene from the background. Nevertheless, many, if not all, the techniques used in 2D environments are also used in this new environment as they help researchers understand the information being presented in order to make more intelligent threshold operations. Various authors in this field are Kumar, Ohya, and Strintzis [33], [30], [17].

Malassiotis and Strintzis [30] propose a 3D gesture recognition based on hand poses. It

includes obtaining 3D information of the scene by illuminating it with a colored pattern, segmenting the arm and the hand, classifying the hand posture, and finally recognizing the 3D gesture.

Finally, Holub et al., [17] use the ZCam to implement an ASL recognition system which starts with skin color detection and depth information and implements an HMM classifier.

1.2.1. The approach in this research

The research presented in this paper uses an approach that is a pose-based recognition using 3D information. The goal is to build a complete solution to the problem of identifying one-hand 3D gestures that include understanding the scene being presented without assuming the user's presence, identifying the user's hand pose and rotation angle in every frame, and constructing the 3D gesture. The static pose approach was selected because one of the visible applications of this work is to integrate it within a multi-touch framework, thus combining the two main gesture environments. This integration requires an ability to rapidly generate new gestures—possible if a wide enough set of poses is included for training. The availability of the ZCam hardware that provides 3D data was the main factor in choosing such input.

Finally, one of the main contributions of this work, with respect to previous work in pose recognition using SVM, is to consider the entire problem without assuming that the user's hand is present in the scene. This would allow the application, once integrated in the multi-touch framework, to run smoothly even when the user leaves or rests his or her arms. Another interesting contribution is the proposed feature set with very little image processing effort required which facilitates the gesture recognition being done in real time.

2. Methods

2.1. Overview of the implemented solution

As previously stated, the selected approach to address the 3D gesture recognition problem is to model the gestures as a sequence of poses. Thus, the gesture recognition problem is reduced to a hand pose recognition problem using 3D information about the scene. Additionally, to completely solve the 3D gesture recognition problem, hand presence/absence and rotation recognition are also considered. This redefines the problem into a triple recognition problem: first, presence/absence of the user's hand; second, the pose of the user's hand; and third, the rotation angle of such pose.

Figure 1 shows a high-level view of the design of the implemented solution. The starting point for the application's data flow is the two 30 FPS streams: one with depth information and one with color video of the scene, produced by the ZCam. Each frame is considered separately for analysis. After some simple image processing operations (described in more detail further on in this document), one or more candidate regions are identified from each frame. Each of these candidate regions is extracted, resized to a 64x64 image, and named “normalized candidate region image.” From each of these normalized candidate region images a novel feature set (also described further on in the document), is extracted and used as input for 3 SVM classifiers. There is one for each of the previously stated subproblems: the user's hand presence/absence, its pose, and its rotation.

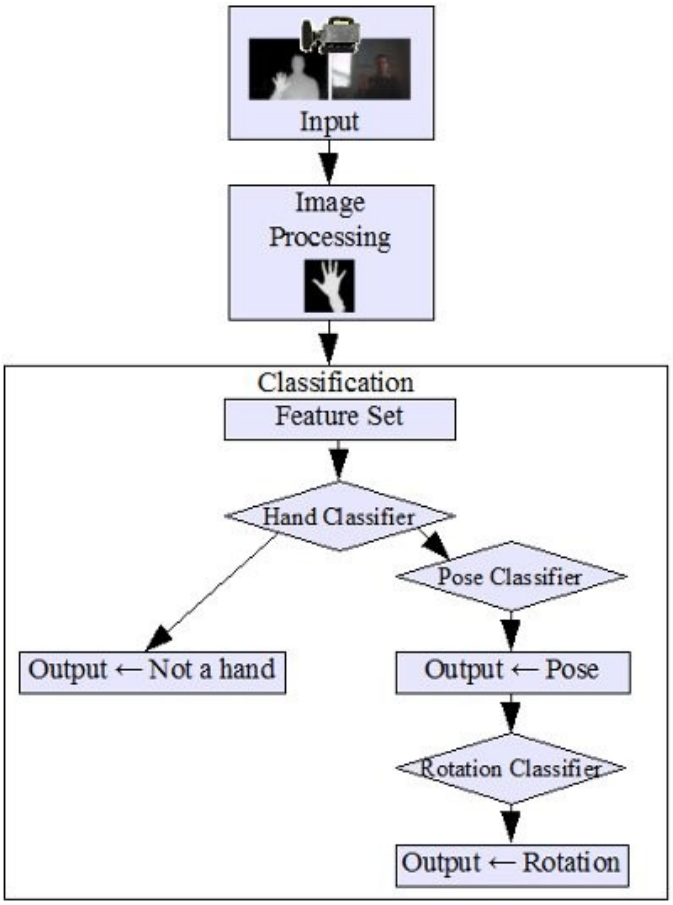

Figure 1. Application's data flow diagram.

2.2. Depth information acquisition and usage

A 3D web cam called ZCam developed by Yahav and 3DV systems [12] was used as the 3D data input device. The ZCam generates infrared light pulses by laser diodes that are reflected by the objects in front of the camera. By capturing such reflections, the device calculates for each pixel the exact distance to the objects in the scene being represented by the pixel. This process is illustrated in Figure 2. In (A), the camera generates infrared pulses that, when reflected in the object, provide the depth information (B).

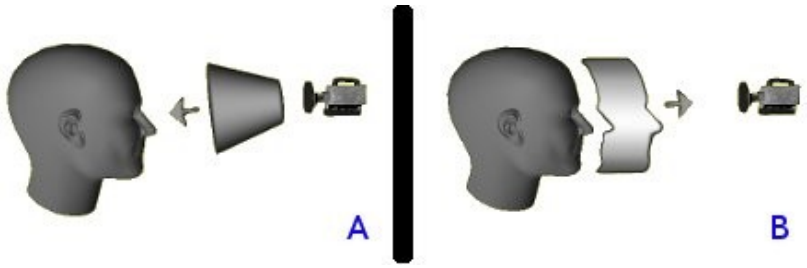

Figure 2. ZCam 3D data acquisition.
Image adapted from Yahav's 3D imaging in the studio. [12]

The ZCam inputs two 320 x 240 images into the system in a rate of 30 FPS (See Figure 5 (A)). The first image is the depth image. This is a gray scale image which reflects the depth information where the brighter the pixel, the closer the object to the camera using 256 levels of gray. The second image is the RGB image. This is a color image similar to the one provided by a normal web cam.

The ZCam builds its depth information of the scene in such a way that the resulting image represents a view of the scene from the perspective of the camera. However, to build a consistent 3D model of the scene, the depth component of the object's representation should be independent of whether the object is in front of the camera or towards the edge of the image. Figure 3 illustrates the perspective problem where Objects 1 and 2 are both aligned. They have the same depth value, but the ZCam places Object 2 farther away (darker) than Object 1 because of the perspective problem.

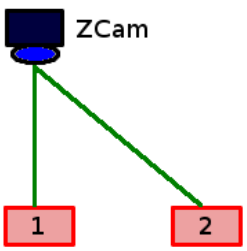

Figure 3. Illustration of the perspective problem where both Objects 1 and 2 are aligned.
They have the same depth value, but the ZCam places Object 2 farther away (darker) than Object 1 because that is the ZCam's perspective.

To solve the perspective problem, the trigonometry shown in Figure 4 was applied, with the result that the pure depth or vertical (z) component for each pixel is obtained by:

$$\sqrt{d^2-((center.x-pixel.x)^2+(center.y-pixel.y)^2)}$$

Where *center.x* and *center.y* are the x and y coordinates of the center of the depth image, *pixel.x* and *pixel.y* are the x and y coordinates of the pixel for which the vertical component is being obtained, and *d* is the actual depth value given by the ZCam for the given pixel.

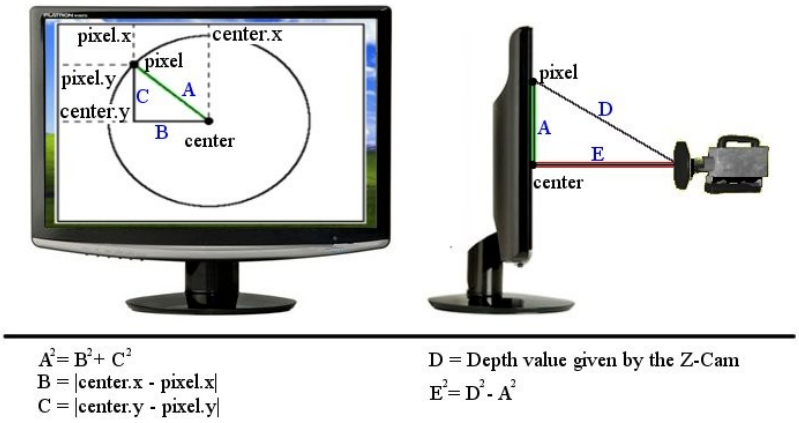

Figure 4. Trigonometry used to solve the perspective problem.

After solving the ZCam's perspective problem and mirroring both the ZCam's original images, a new version of both frames is obtained as shown in Figure 5 (B). Note how the background noise in both left and right sides of the new image are lighter than in the original image—that is a result of fixing the perspective.

Assuming that if the user's hand is present in the scene it will be closer to the camera than the rest of the body, the approach was to identify the pixel closest to the camera (the brighter

pixel) and threshold the depth image so it would only consider a small window of depth starting from the closest point (20 levels of gray). The new depth image would only contain blobs for the closest objects. These blobs are extracted and resized into a new 64x64 image shown in Figure 5 (C) which in turn will be the input from where the feature set will be extracted and fed to the classifiers.

Figure 5. Image processing evolution.
A: Original images generated by the ZCam.
B: Resulting images after being mirrored and applied the perspective problem fix.
C: Resulting image after thresholding 20 levels of gray from the closest point, extracting the blob, and resizing into a new 64x64 image.

2.3. The feature set

When designing a feature set, the goal is to identify a set of characteristics that will separate samples of one class from samples of another. A good feature set is one that correctly separates samples of different classes. However, as the set of characteristics grows larger or more complex, the required effort to process them also increases. Therefore, if two

feature sets achieve the same sample separation, the feature set with simpler characteristics is preferable over the one with more complex characteristics. In the case of this research, the samples to be separated are samples of the different hand poses, samples of the different rotation poses, and samples where the hand is present and where it is not. Using the input of the normalized candidate region image, a novel feature set based on simple characteristics is proposed to model the sample universe and to separate samples of different classes. The main idea is to detect alternations or "jumps" between black (void) and some level of gray (hand) using the number and size of the jumps as the chosen characteristics that form the feature set.

Alternations between black and gray characterize how many fingers the user shows and whether they are separated. For example, if the image is an open hand with the fingers pointing up, there should be several small jumps in the upper rows of the image. If the image is a closed fist, then the number of alternations should be quite low as the image would be similar to a solid block in the center of the screen. Moreover, if the image is a single pointing finger, then there will be a smaller solid block (the finger) and a bigger one (the rest of the hand) in the image. Figure 6 shows the open hand and the pointing hand examples where the jumps of the highlighted row (red) are show in blue and towards the right of the image.

The same concept of searching for jumps is used to capture the 3D information of the image. Given that the closer the object is to the camera, the brighter the pixel that represents that object, the brightness levels can be used to find the jumps in depth plane. So, for example, in the case of a hand pointing with one finger to the camera, there would be a big jump for the finger and a smaller jump for the rest of the image. Figure 6 shows two images of hands pointing to the camera: in the first image there is an open hand where only the index and the little finger are extended; in the second image, the hand is closed and pointing only

with one finger. The corresponding jumps are again drawn in blue towards the right of the image.

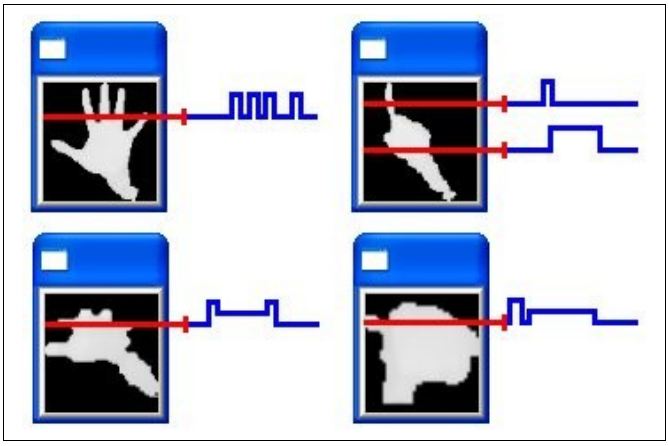

Figure 6. Hand characterization based on black and gray alternations.
In the upper images, jumps are used to distinguish between an open hand and a pointing hand. In the lower images, jumps based on brightness intensity are used to distinguish between two hands pointing to the camera.

To implement the previously described feature set, an extension of the algorithm is used for handwriting recognition [13], [41]. The handwriting recognition algorithm starts from a 64x64 image and divides it into 64 8x8 windows. Then for each window, it searches for vertical, horizontal, and diagonal patterns. In this work, the normalized candidate region image is divided into 64 8x8 windows as shown in Figure 7, where each of these windows is represented by an integer in the feature set. Note that using 8x8 windows in a 64x64 image results in a 64 elements feature set, so using smaller windows would help obtain more detailed information of the candidate region image, but would also require a bigger feature set, which in turn would require more processing effort. Each of these 8x8 windows are divided again into four 4x4 smaller windows, where each of these last windows are represented by a flag in a 4-bit number. The flag of the 4x4 window will be turned on (1) if at least one of the 4x4 bits is not 0 (a bit in 0 means black or absence of object in front of the

camera). It will be 0 otherwise. So for each of the 8x8 windows, there will be a number between 0 (there is no object in the 8x8 window) and 15 (the window is completely covered) which represents the presence or absence of an object in that window. Figure 7 shows an example of a normalized candidate region image divided into the 64 8x8 windows, and then shows how three of these windows are divided again into four 4x4 inner windows that in turn are used as flags in a 4-bit number as previously explained.

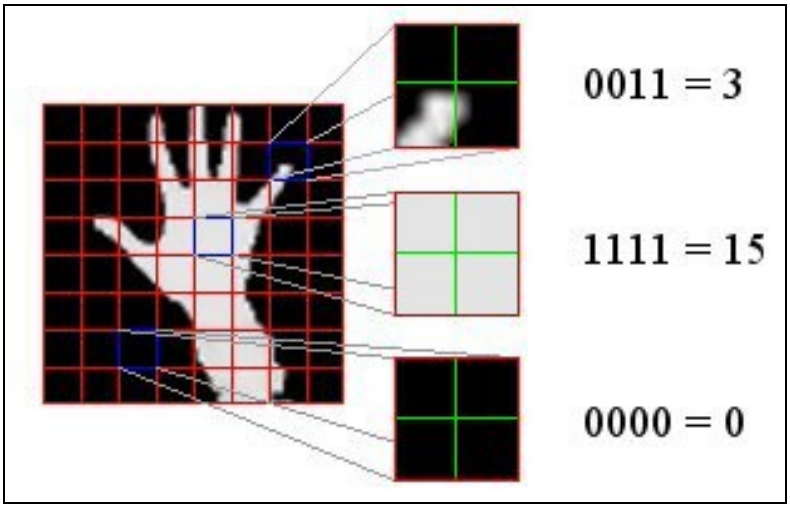

Figure 7. Feature extraction from normalized candidate region image.
This figure shows the extracting back and gray alternations process by first dividing the image into 64 8x8 windows and then dividing each of these windows again into four 4x4 inner windows and using these last windows as flags in a 4-bit number. In the image, three of the 64 8x8 windows are highlighted to show how they are divided again into 4x4 inner windows.

The depth information is also relative to the 8x8 windows. Taking advantage of the fact that the normalized candidate region image has a fixed depth (20 levels of gray), the depth window is divided into four equidistant regions numbered from one to four, where one is the closest region (with larger depth values) and four is the farthest region (with lower depth values) as shown in Figure 8. Then for each of the 8x8 windows, the pixel with the highest depth value is considered and set into one of the previous four regions. The depth region

number for the closest pixel of the window would be the rightmost digit of the integer describing the window in the feature set.

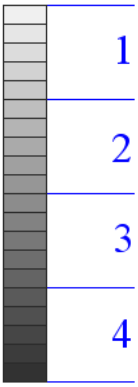

Figure 8. 20 levels of gray divided into regions.
Dividing the 20 levels of gray into four equal regions to characterize the depth level of each of 8x8 windows in which the image was divided.

So up to this point, the feature set is composed by 64 3-digit numbers, one for each of the 8x8 windows, where each of these numbers contains information about the black gray alternations as well as brightness or depth alternations.

There is a 65th integer included in the feature set. It is a flag that would take value 1 if the TDV hand-tracking tool provided by the ZCam SDK recognizes a hand, and value 0 otherwise. R. Jordan-Osorio and Sukhoy [28] did a survey analyzing the accuracy of the TDV hand-tracking tool with the following results:

Gesture		Accuracy % for left hand	Accuracy % for right hand
Fist		97%	97%
thumb		95%	96%
index		96%	96%
index and thumb		85%	94%
index and middle		89%	93%
index, middle and thumb		72%	88%
five fingers		70%	75%

Table 1. Accuracy of the TDV hand tracking tool provided by the ZCam SDK. From *Proteins visualization control using hand gestures* [28].

Notice that in all cases, the hand is placed with the fingers pointing up. The accuracy drops when the hand is rotated to other positions.

2.4. The classifiers

Three classifiers were implemented. The first classifier (called “hand classifier”) decides if the image being analyzed corresponds to a hand or not. If the image is classified to correspond to a hand, then the second classifier (called “pose classifier”), categorizes the hand into one of the predefined poses or the “other/undefined pose.” Finally, the third classifier (called “rotation classifier”) defines the rotation of the hand as it best approximates one of the predefined rotation angles.

Both the pose classifier and the rotation classifier can be trained for a different number of poses and different rotation angles. Neither the algorithm nor the application restricts the number of poses or rotations, and there is also no restriction on including any specific pose or

rotation angle. In this case, six[1] different poses were predefined: fist, open hand with separate fingers, open hand with fingers together, pointing hand (with one and two fingers), showing two fingers, and showing three fingers. Regarding the rotation, five different rotation angles or positions were predefined: left, up-left, up, up-right, and right. See Figure 9 for an illustration of the poses and rotation positions.

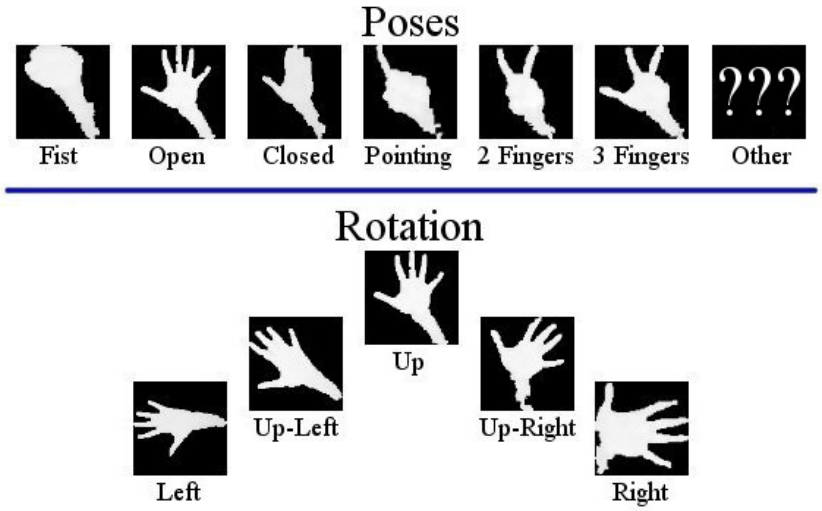

Figure 9. Selected poses and rotation positions.

All three classifiers were implemented as Support Vector Machines (SVM) trained through the Sequential Minimal Optimization (SMO) algorithm using Weka's [16] API (Application programming interface).

2.4.1. Support Vector Machines and Sequential Minimal Optimization algorithms

Support Vector Machines classifiers (invented by Vapnik in 1979), try to find an hyperplane that separates samples of the different classes, maximizing the distance between the decision boundary and any of the samples. The minimum of these distances between the

1 6 poses and 5 rotation positions were predefined as the combinations of them would cover an interesting number of possible gestures. If needed more poses and more rotation positions could be added.

decision boundary and each sample is called the margin [8].

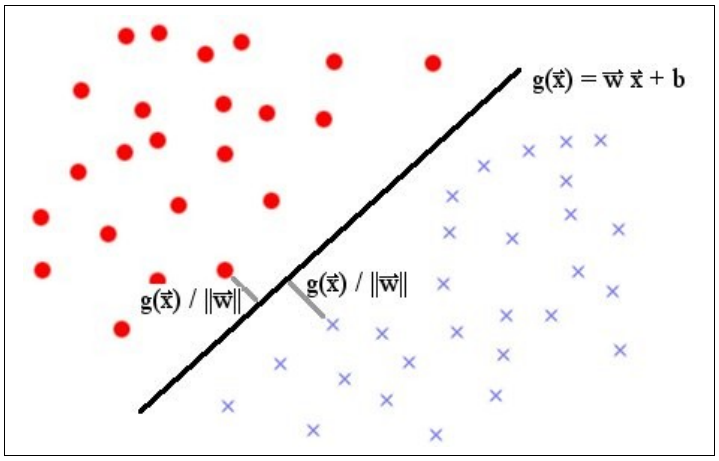

Figure 10. Support Vector decision boundary.

Assuming the sample set is linearly separable, the decision boundary is obtained through:

$$g(\vec{x}) = \vec{w}^{t}\vec{x} + b$$

Where w is the weights vector, x is the samples features (input) vector, and b is a constant. The algebraic distance of a point (sample) to the decision boundary and the margin of the training set are:

$$distance_i = \frac{g(\vec{x})}{||\vec{w}||}$$
$$margin = min_i \frac{t_i g(\vec{x}_i)}{||\vec{w}||}$$

Where $t_i \in \{-1, +1\}$ and if a data sample is correctly classified, then $t_i\, g(\boldsymbol{x}_i) > 0$. Given that the definition of an hyperplane does not change when rescaled and that the margin also is not

affected by rescaling, it is possible to rescale so that $min_i\ t_i\ g(\mathbf{x}_i) = 1$, then the margin is now equal to 1/||w||, so minimizing ||w|| would maximize the margin. Then the resulting *quadratic programming* (QP) minimizing problem is:

$$\begin{aligned} &\textit{Minimize} \\ &\quad \frac{\vec{w}^t \vec{w}}{2} \\ &\textit{subject to} \\ &\quad t_i(\vec{w}^t \vec{x} + b) \geq 1 \quad i = 1, \ldots, n \end{aligned}$$

Through Lagrangian theory, the dual optimization problem for the previous primal is:

$$\begin{aligned} &\textit{Maximize} \\ &\quad L_d(\vec{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j (x_i \cdot x_j) \\ &\textit{subject to} \\ &\quad \sum_i \alpha_i t_i = 0 \\ &\quad \alpha_i \geq 0 \quad \forall i \end{aligned}$$

The Karush-Kuhn-Tucker (KKT) conditions for the previous problems optimal solutions are:

$$\begin{aligned} &\vec{w} = \sum \alpha_i t_i \vec{x}_i \\ &\sum \alpha_i t_i = 0 \\ &\alpha_i \geq 0 \\ &t_i(\vec{w}\,\vec{x} + b) - 1 \geq 0 \\ &\alpha_i [t_i(\vec{w}\,\vec{x} + b) - 1] = 0 \end{aligned}$$

Then, only the samples x_i for which $t_i (w_i x_i + b) = 1$ can have $\alpha_i \neq 0$. These samples are the support vectors as they are the closest samples to the decision boundary and they contain all the necessary information to reconstruct the decision boundary hyperplane.

In case the samples are not linearly separable, "slack" variables are introduced in the primary problem to relax the constraint that all training data must be correctly classified. The slack variable ξ_i represents how much the sample x_i fails to respect the margin of 1 from the deciding boundary. After introducing the slack variables the primal quadratic programming problem definition is:

$$
\begin{aligned}
&\textit{Minimize} \\
&\quad \frac{\vec{w}^t \vec{w}}{2} + C \sum_i \xi_i \\
&\textit{subject to} \\
&\quad t_i (\vec{w}^t \vec{x} + b) \geq 1 - \xi_i \quad i = 1, \ldots, n \\
&\quad \xi_i \geq 0 \quad i = 1, \ldots, n
\end{aligned}
$$

where the ξ_i is the slack variable and C is a constant penalty for usage of the slack component. The dual problem with slack variables would be:

$$
\begin{aligned}
&\textit{Maximize} \\
&\quad L_d(\vec{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j (x_i \cdot x_j) \\
&\textit{subject to} \\
&\quad \sum_i \alpha_i t_i = 0 \\
&\quad 0 \leq \alpha_i \leq C \quad \forall i
\end{aligned}
$$

Again the support vectors are the only samples for which $t_i (w_i x_i + b) = 1 - \xi_i$ meaning

that $\alpha_i \neq 0$.

In many cases, classes are not linearly separable. A possible solution is to use non-linear transformations into a feature space where the data becomes separable. The problem here is dealing with high dimensional feature spaces and the high risk of overfitting. To avoid the problems of working with high dimensional feature spaces, kernel functions can be used as long as the data points only appear inside dot products. A kernel is a function which returns the result of the dot product of the images of the samples in the new feature space, even when the transformation function Φ is not known:

$$K(\vec{x}_1, \vec{x}_2) = \Phi(\vec{x}_1)^t \Phi(\vec{x}_2)$$

Given that in the dual representation of the problem, samples only appear in a dot product, so kernel substitutions can be done. The final dual representation is:

$$\begin{aligned} &\textit{Maximize} \\ &\quad L_d(\vec{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j K(x_i, x_j) \\ &\textit{subject to} \\ &\quad \sum_i \alpha_i t_i = 0 \\ &\quad 0 \leq \alpha_i \leq C \quad \forall i \end{aligned}$$

For the particular case of this work, a polynomial kernel was used based on Weka's implementation:

$$K(x_1, x_2) = (x_1 \cdot x_2)^d$$

SMO is an algorithm for training support vector machines proposed by Platt [27]. The main problem when training SVM is the size of the QP problem that involves a matrix of size square to the number of training samples. There are several methods to reduce the size of the matrix, but they still require solving the QP problem numerically.

The SMO algorithm takes advantage of the theorem proved by Osuna et al., [10] which proves that a large QP problem can be reduced to smaller QP sub-problems as long as at least one sample that is not optimized (violates the KKT conditions) is added to the previous sub-problem. Based on the previous theorem, the SMO algorithm would solve the smallest possible QP problem, which in the case of SVM includes two Lagrangian multipliers. The greatest advantage of the SMO training method is that it only includes two Lagrangian multipliers per step, so it can resolve the QP problem analytically without performing the numerical optimization.

There are two main parts to the SMO training algorithm, solving the two Lagrangian multipliers QP problem and identifying which multipliers to use.

First, identify the two Lagrangian multipliers to be solved. The algorithm first computes the constraints on those multipliers and then obtains the constrained maximum. The constraint $0 \leq \alpha_i \leq C$ restricts the multipliers to lie within a box and the constraint $\sum_i \alpha_i t_i$ places the multipliers within a diagonal line. So both constraints together restrict the multipliers to lie within a well-defined segment. The SMO algorithm then calculates the maximum in the defined segment and moves the Lagrangian multipliers to that point. (Special cases, such as when both ends of the segment have the same objective value, are considered in the original paper).

Second, identify which of the Lagrangian multipliers should be used. This is done through two heuristics: one to obtain the first multiplier and the other to obtain the second one. To select the first multiplier, the algorithm goes through all samples once and identifies those that do not satisfy the KKT conditions. Then it completes a second loop only through those samples where the multiplier is neither 0 nor C (called the "non-bound samples"), again identifying those samples that do not satisfy the KKT conditions. Samples that do not satisfy the KKT conditions are eligible for optimization. The algorithm selects the non-bound samples for optimization first, as they are more likely to change during the optimization process. Once all non-bounding samples are optimized, the algorithm moves back to the rest of the samples and repeats until finished (the process can include several alternating loops over non-bound samples and total set of samples). The second multiplier is selected to maximize the size of the step taken during the optimization process by comparing the errors of each sample and choosing the sample that has the biggest error difference with the first sample.

3. Classifiers performance results

A tool was developed to generate both the training set and an independent testing set. The generation of both the training set and the independent testing set is an iterative process where each cycle consists in recording frames, performing the image processing operations described in the previous section, classifying the resulting image, and generating one training/testing file for each classifier. First, in each cycle, the training tool records 100 frames and performs the required image processing to each frame. When each frame is ready, the user training the tool must classify the resulting image. This is done by pressing predefined keys (H for hand, N for Not Hand, F for fist, and so on), that automatically generates entries for three training files, one for each classifier. For this research, two training set were created and labeled, one by the author with 4,956 samples for the hand classifier, 8,575 for the pose classifier, and 3,113 for the rotation classifier and another one by an independent non-computer-science undergraduate user who did not participate in any other part of the development with 8,451 samples for the hand classifier, 8,098 for the pose classifier and 8367 for the rotation classifier. The distribution among classes is shown in Table 2. Note that the load among classes was not balanced for the hand and pose classifiers. This could cause the classifiers to prefer (classify more frequently) those classes with more samples reducing the classifier's accuracy. To avoid such situations, Weka's filter SMOTE was used to balance the sample loads. The SMOTE filter applies the Synthetic Minority Oversampling TEchnique (SMOTE) introduced by Chawla et al. [26]. It over-samples the minority classes by introducing new samples in the line segments joining the original sample with its nearest neighbors. As a result of applying the filter, the new training files were

balanced, also shown in Table 2.

Class	Author's training		Independent user's training	
	Before Balancing	After Balancing (SMOTE)	Before Balancing	After Balancing (SMOTE)
Hand Classifier				
No Hand	1,331	**2,662**	8,140	8,140
Hand	3,625	3,625	311	**1,247**
Pose Classifier				
Fist	556	**1,112**	555	**1,114**
Open	1,236	1,236	1,631	1,631
Closed	1,713	1,713	1,351	1,351
Pointing	1,953	1,953	1,298	1,298
2 Fingers	1,186	1,186	1,542	1,542
3 Fingers	1,192	1,192	1,433	1,433
Other	731	731	288	**573**
Rotation Classifier				
Left	593	593	1,611	1,611
Up-Left	559	559	1,387	1,387
Up	644	644	2,499	2,499
Up-Right	663	663	1,406	1,406
Right	654	654	1,464	1,464

Table 2. Training load for each classifier.

Additionally, an independent testing set was generated by the author in a similar manner: frames were recorded, classified, and stored in testing files, resulting in 1,491 (255, 1,236) test cases for the hand classifier, 1,237 (105, 211, 248, 236, 169, 125, 143) for the pose classifier, and 988 (75, 288, 433, 135, 57) for the rotation classifier. (There is no point in balancing the testing set; in fact, balancing the testing set would distort the test results.) The process of generating training samples and generating testing samples are similar. But when

generating the training samples, the training case (the hands pose and rotation) was defined before recording—as the user generated the testing samples, he or she would move his or her hand arbitrarily, as any user would do while using the application to perform gestures for the camera. This was done to generate more transitions and more natural poses in the testing set. Both training sets were tested against the same independent testing set to measure the intercoder reliability among both training sets.

There are several metrics to evaluate the performance of a classifier. True positive rate (TPR) or sensitivity, false positive rate (FPR) or 1 – specificity, and correlation coefficient (CC) are commonly used for such purpose and they are defined as follows:

$$TPR = \frac{TP}{TP + FN} \quad FPR = \frac{FP}{TN + FP}$$

$$CC = \frac{TP \times TN - FP \times FN}{\sqrt{(TN + FN)(TN + FP)(TP + FN)(TP + FP)}}$$

Although these are commonly-used metrics to evaluate classifiers, the evaluation is done over a single predefined classification threshold which, if changed, could drastically alter classifier behavior. The receiver operating characteristic (ROC) curve evaluates a classifier over all possible thresholds. The ROC curve is a plot between the true positive rate versus the false positive rate changing the classifier's threshold. The area under the ROC curve (AUC) is used as a single number summary of the ROC curve, which could be used as a measure of the accuracy of the classifiers (Huang and Ling [15]). A perfect classifier would have an AUC = 1, whereas the worst possible classifier (one that cannot discriminate) would have an

AUC = 0.5. Figure 11 illustrates the ROC curve of three classifiers and shows how the best classifier is the one with AUC closest to 1.

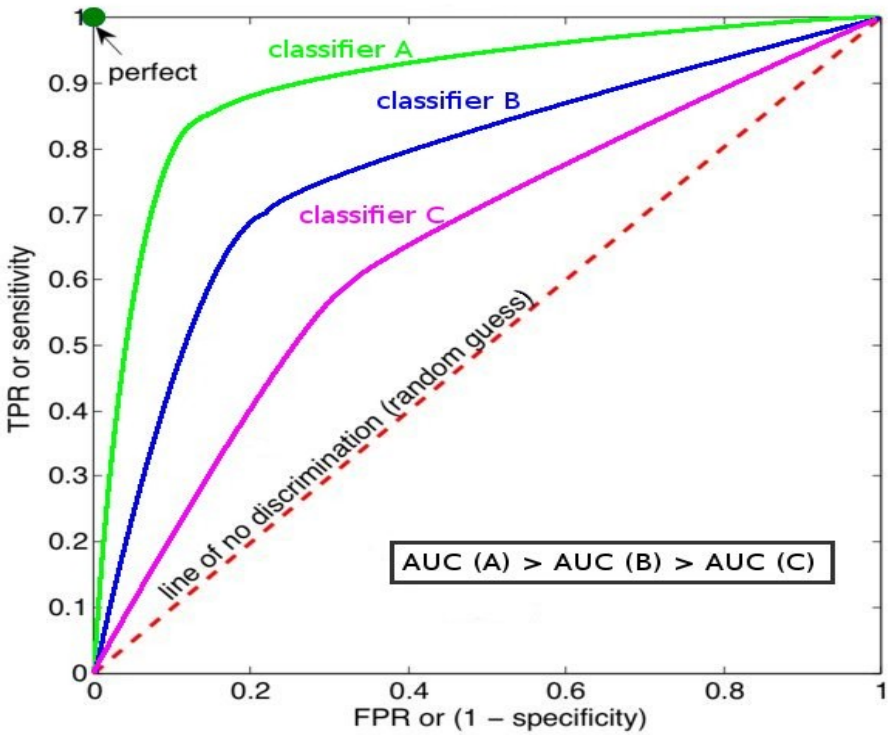

Figure 11. ROC and AUC illustration.

The performance of the three classifiers used in this work (hand, pose, and rotation), according to the previously mentioned criteria and for both training sets are summarized in Table 3.

	Using author's training set				Using independent user's training set			
	TPR	**FPR**	**CC**	**AUC**	**TPR**	**FPR**	**CC**	**AUC**
	Hand Classifier							
No Hand	0.867	0.054	0.775	**0.962**	0.404	0.017	0.528	**0.875**
Hand	0.946	0.133	0.775	**0.944**	0.983	0.596	0.528	**0.857**
	Pose Classifier							
Fist	0.810	0.036	0.713	**0.970**	0.876	0.080	0.625	**0.956**
Open	0.768	0.065	0.680	**0.931**	0.308	0.038	0.366	**0.796**
Close	0.532	0.043	0.562	**0.865**	0.343	0.111	0.254	**0.717**
Pointing	0.568	0.042	0.591	**0.926**	0.771	0.142	0.562	**0.871**
2 Fingers	0.704	0.084	0.568	**0.919**	0.533	0.156	0.318	**0.761**
3 Fingers	0.744	0.070	0.588	**0.884**	0.416	0.094	0.291	**0.719**
Other	0.594	0.060	0.522	**0.865**	0.007	0.013	-0.017	**0.609**
	Rotation Classifier							
Left	0.920	0.010	0.894	**0.996**	0.880	0.011	0.864	**0.975**
Up-Left	0.747	0.023	0.777	**0.942**	0.705	0.029	0.735	**0.939**
Up	0.968	0.209	0.755	**0.880**	0.822	0.164	0.656	**0.895**
Up-Right	0.615	0.029	0.644	**0.954**	0.874	0.106	0.648	**0.939**
Right	0.596	0.002	0.740	**0.970**	0.456	0.009	0.572	**0.975**

Table 3. Results for the three classifiers when implemented as SVM-SMO with polynomial kernel of degree 1.

The plotted ROC curves for the three classifiers (hand, pose and rotation) when using the author's training sets are shown in Figure 12, Figure 13 and Figure 14 respectively.

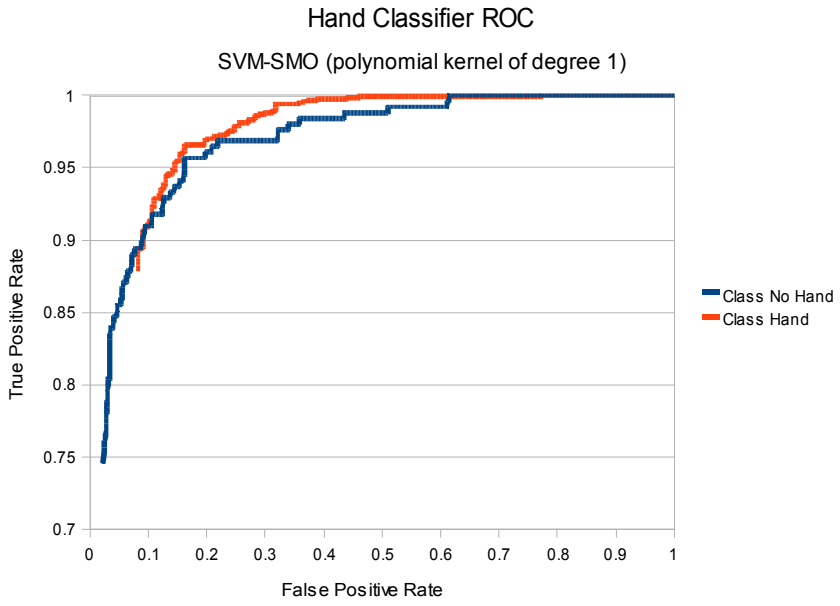

Figure 12. ROC curves for the hand classifier trained by the author.

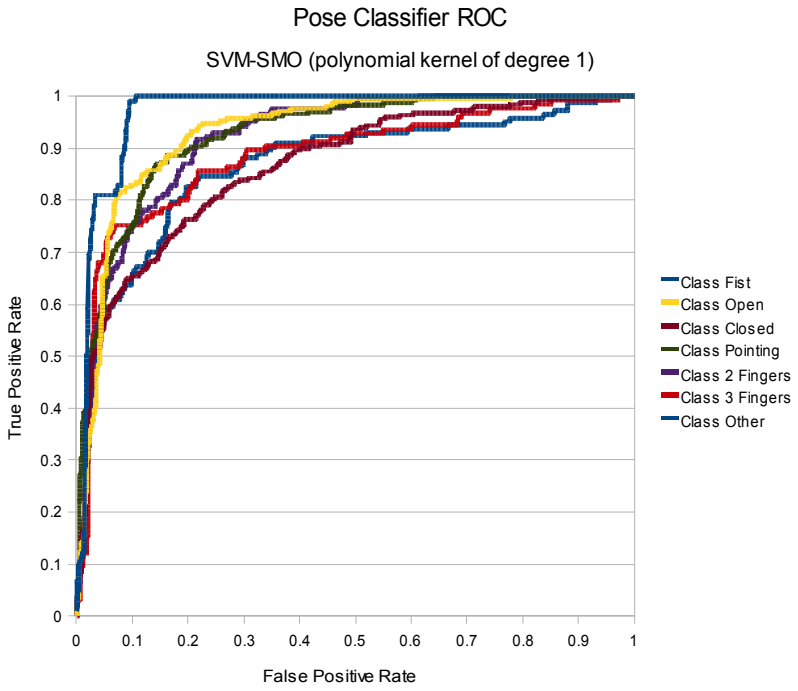

Figure 13. ROC curves for the pose classifier trained by the author.

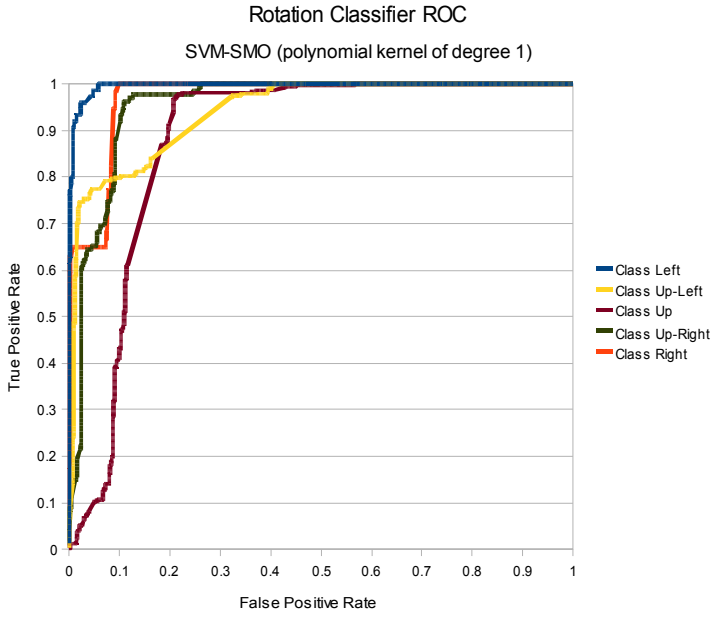

Figure 14. ROC curves for the pose classifier trained by the author.

The plotted ROC curves for the three classifiers (hand, pose and rotation) when using the

independent user's training sets are shown in Figure 15, Figure 16 and Figure 17 respectively.

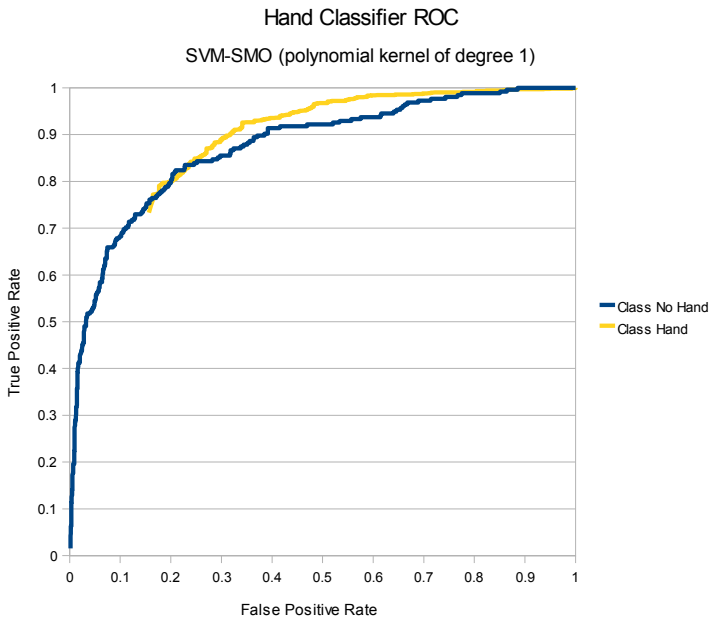

Figure 15. ROC curves for the hand classifier trained by the independent user.

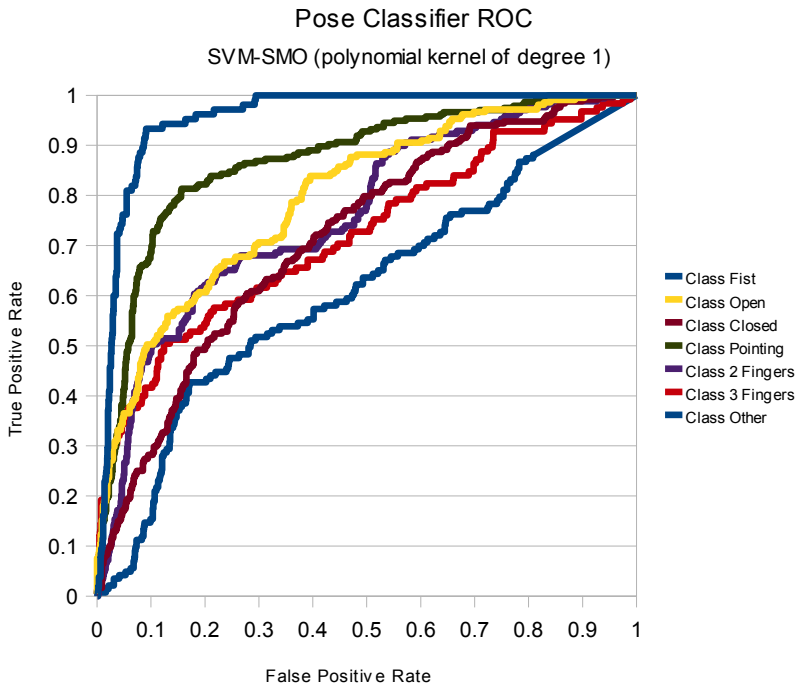

Figure 16. ROC curves for the pose classifier trained by the independent user.

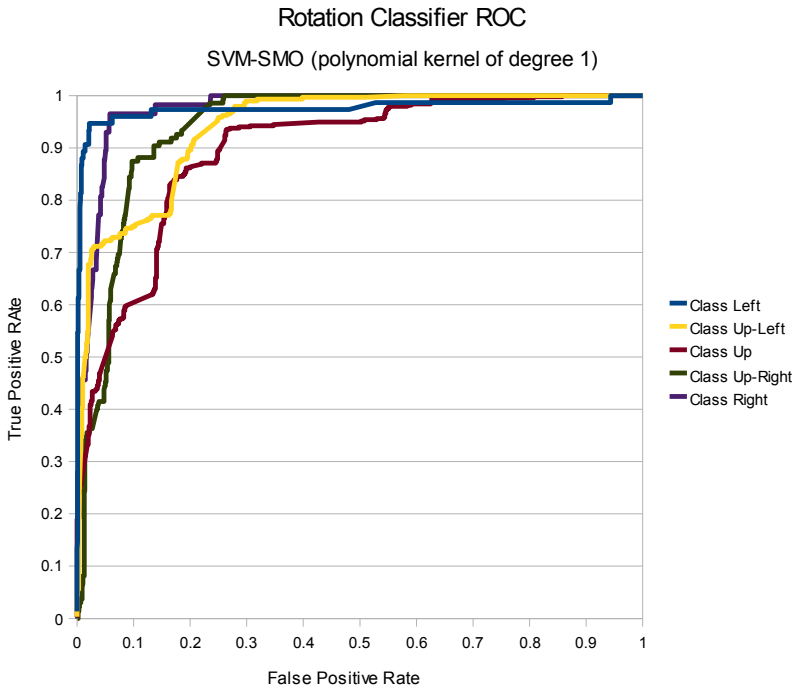

Figure 17. ROC curves for the rotation classifier trained by the independent user

Understanding that the AUC metric indicates the probability of a given class to be actually positive when classified as such, averages values, over all classes, of 93% when using the author's training set and 85% when using the independent user's training indicate that the SVM-SMO classifier is a reliable classifier for all three problems being addressed. However, when using the independent user's training set there were some low AUC values that cause the classifier to be less reliable regarding those classes. However, to obtain a value of intercoder reliability, an extension of the Holsti's [14] method was used. Instead of counting the number of decisions upon which the two trainers agree, the correlation coefficient obtained after testing both training sets against the same testing set was used. So the intercoder reliability measure was obtained by:

$$\text{Average over all classes } \{1 - |CC_{author} - CC_{independent\ user}|\}$$

The resulting value is: **80%**, which exceeds 70%, the minimum requirement for reliability. This reliability demonstrates that the quality of the training set does not depend strongly on who does the training. Furthermore, the lack of special training for the independent user (an undergraduate with no previous exposure to the project) indicates that there is no need of a deep understanding of how the system works to successfully label a high quality training set.

After a closer look at the rotation classifier, it seems that the up class has a noticeable lower curve than the other classes. This is because of the difficulty (even during the training phase) to clearly distinguish between up versus up-left and up versus up-right. A similar problem occurs with the classes left and right respectively, but it does not seem to have the same effect. A possible solution to this problem is to calculate an approximation to the rotation angle instead of using classifiers. This possibility is included within the future work section of this paper.

Although the previous results are promising, it is interesting to compare them with the results that other classification algorithms have produced with the same training and testing sets, verifying that the SVM-SMO algorithm is the best for the job. Only the author's training set was used for these comparisons. So the previous results were compared with other commonly used classifier implementations. Naive Bayes, J48 decision tree, Adaboost with J48 tree, and Adaboost with ID3 tree were used.

The Naive Bayes classifier performs its classification by obtaining the posterior probability of each class given a set of evidence:

$$\omega_{NB} = arg\,max_{\omega_j} P(\omega_j) \prod_i P(x_i|\omega_j)$$

A key assumption of the Naive Bayes classifier is that all features in the feature space are statistically independent (this is why the classifier is called “naïve”). Although this assumption seems pretty strong, the Naive Bayes classier performs well in various settings.

$$P(x_{1,} x_{2,} \ldots, x_n|\omega_j) = \prod_i P(x_i|\omega_j)$$

The J48 decision tree is Weka's implementation of the C4.5 model from J. R. Quinlan [19]. Basically, decision trees try to identify the feature that best discriminates between classes and creates a branch for each of the possible values of such feature. If necessary, each branch performs a similar selection until all samples of the branch belong to the same class.

Adaboost algorithms ensemble different classifiers to obtain a better overall performance. The goal of ensemble learning classifiers is to eliminate individual errors by averaging between several classifiers; in general the averaging is done through weighted vote. Boosting algorithms manipulate the training set by setting different weights to the samples of the training set. Then the inner classifiers are trained with the different weighted data sets, resulting in different base classifiers. Here, the base classification algorithms used were the J48 decision tree and the ID3 decision tree. Decision trees were selected as the base classifiers because they are relatively fast to train, and small changes in the data set have a great affect on the resulting tree.

Table 4 compares the performance of the three classifiers (hand, pose, and rotation), trained with the author's training set, based on the predefined metrics implementing the different classification algorithms. (The values in the table are average values over all classes.)

	TPR	FPR	CC	AUC
	Hand Classifier			
SVM-SMO	0.932	0.120	0.775	0.947
Naive Bayes	0.939	0.221	0.774	0.952
J48 decision tree	0.893	0.212	0.647	0.885
Adaboost-J48	0.946	0.154	**0.808**	0.966
Adaboost-ID3	0.889	0.076	0.699	**0.968**
	Pose Classifier			
SVM-SMO	0.655	0.057	**0.603**	**0.906**
Naive Bayes	0.487	0.086	0.431	0.850
J48 decision tree	0.490	0.091	0.399	0.775
Adaboost-J48	0.643	0.064	0.579	0.891
Adaboost-ID3	0.333	0.075	0.317	0.775
	Rotation Classifier			
SVM-SMO	0.830	0.103	0.762	0.922
Naive Bayes	0.840	0.097	**0.798**	**0.970**
J48 decision tree	0.748	0.130	0.621	0.856
Adaboost-J48	0.804	0.108	0.721	0.909
Adaboost-ID3	0.738	0.181	0.673	0.912

Table 4. Performance over the three classifiers of different classifiers algorithms implementation.

The plot of the ROC curves comparing the performance of the different classifier implementations for each of the classifiers in one example class are shown in Figure 18, Figure 19, and Figure 20.

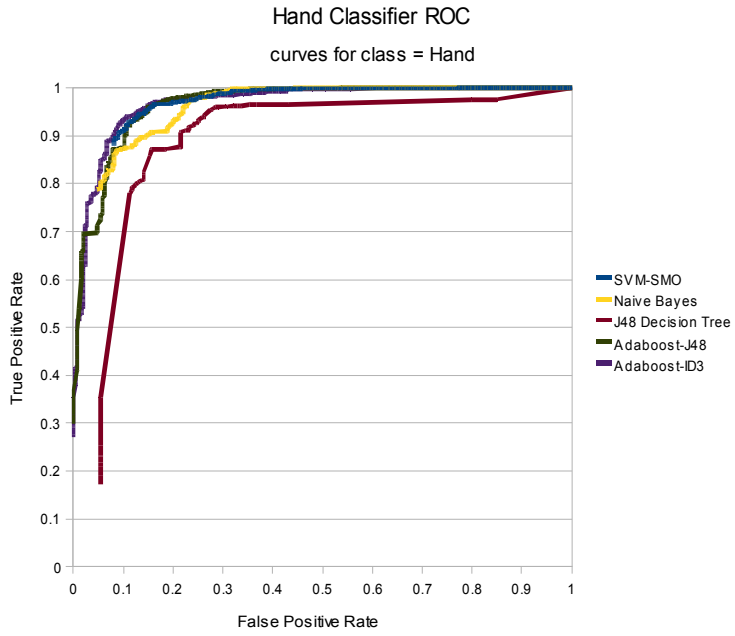

Figure 18. ROC curves for the hand class in the hand classifier for the different classification algorithms.

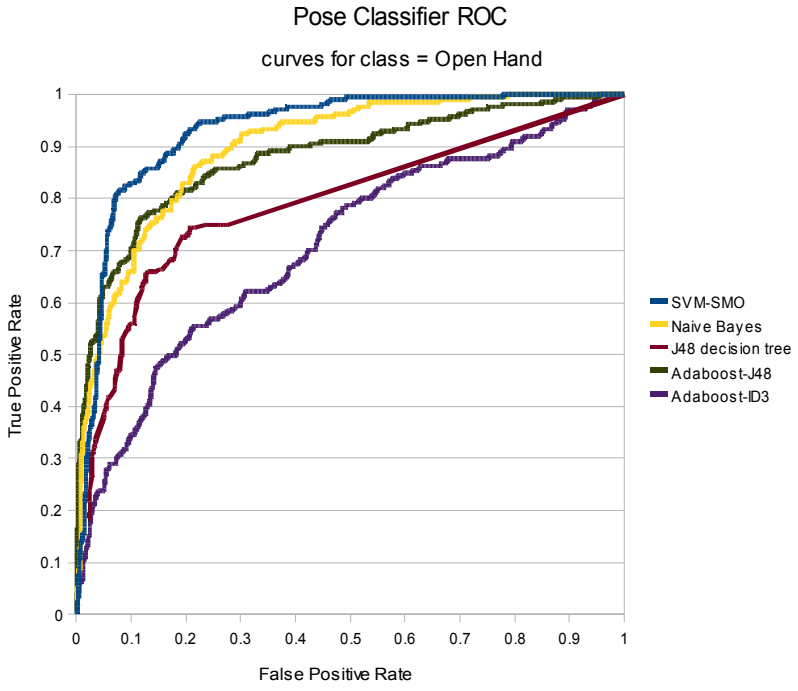

Figure 19. ROC curves for the open hand class in the pose classifier for the different classification algorithms.

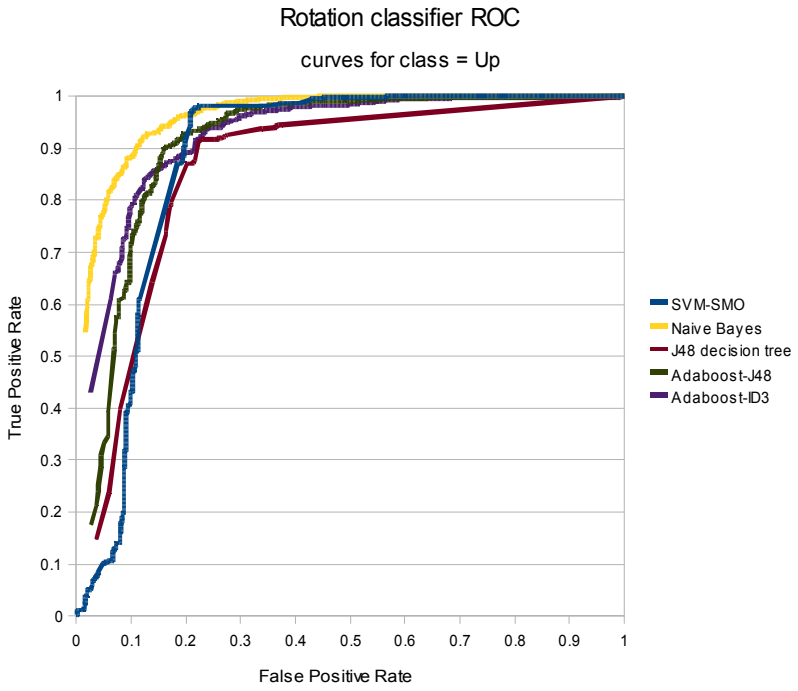

Figure 20. ROC curves for the up class in the rotation classifier for the different classification algorithms.

When reviewing the previous results for the hand classifier, it can easily be concluded that, with the exception of the J48 tree algorithm, all classifiers had similar performances between 94% and 97%. This means that the selected feature set is a good representation of this particular model. The case is not the same for the pose classifier where the SVM-SMO classifier clearly outperforms all other algorithms. Finally, in the case of the rotation classifier, the Naive Bayes algorithm seems to be the best algorithm for this classifier, with SVM-SMO coming in second place.

From these results, and considering that in both the hand and the rotation classifiers the top classifiers performed similarly, the overall best algorithm for all three classifiers is the SVM-SMO.

During the overview of the SVM classifiers, it was stated that the kernel used by the

classifier plays a key role in the performance of the classifier. So to test if any improvement can be obtained, the same three classifiers were implemented and tested using different kernels, again only the author's training set was used for these experiments. Up to this point, a polynomial kernel of degree 1 was used. Then, to evaluate the impact of changing the kernel, polynomial kernels of degree 2 and 3 were chosen. Additionally, an instance of the Radial Basis Function (RBF) kernel, which is a Gaussian-based kernel, was also selected for the comparative study. The RBF kernel is:

$$K(x, y) = e^{-\left(\frac{\|x-y\|^2}{2\sigma^2}\right)} = e^{-(gamma\,\|x-y\|^2)}$$

For this work the gamma parameter was set to 0.001.

Table 5 compares the performance of the three classifiers (hand, pose, and rotation) implemented with the SVM-SMO algorithms using the different kernels previously mentioned. (Again, the values in the table are average values over all classes.) Although in general, the more complex kernels outperform the selected polynomial 1-degree kernel, the processing time to train the classifiers with these more complex kernels was considerably longer (sometimes more than five times longer). However, the actual classification processing time will not be affected in the same way. So the extra effort is just a one time offline situation.

SVM-SMO	TPR	FPR	CC	AUC
	Hand Classifier			
Polynomial Kernel degree 1	0.932	0.120	0.775	0.947
Polynomial Kernel degree 2	0.968	0.084	0.886	0.963
Polynomial Kernel degree 3	0.973	0.077	**0.902**	0.968
Radial Basis Function Kernel	0.956	0.099	0.846	**0.983**
	Pose Classifier			
Polynomial Kernel degree 1	0.655	0.057	0.603	0.906
Polynomial Kernel degree 2	0.747	0.039	0.710	0.932
Polynomial Kernel degree 3	0.766	0.037	**0.729**	0.938
Radial Basis Function Kernel	0.730	0.048	0.683	**0.946**
	Rotation Classifier			
Polynomial Kernel degree 1	0.830	0.103	0.762	0.922
Polynomial Kernel degree 2	0.840	0.100	0.784	0.925
Polynomial Kernel degree 3	0.843	0.098	**0.794**	**0.927**
Radial Basis Function Kernel	0.826	0.105	0.757	0.918

Table 5. Performance over the three classifiers of the SMO-SVM algorithm using different kernels.

Plots of the ROC curves comparing the different kernels of some sample classes are shown in Figure 21, Figure 22, and Figure 23.

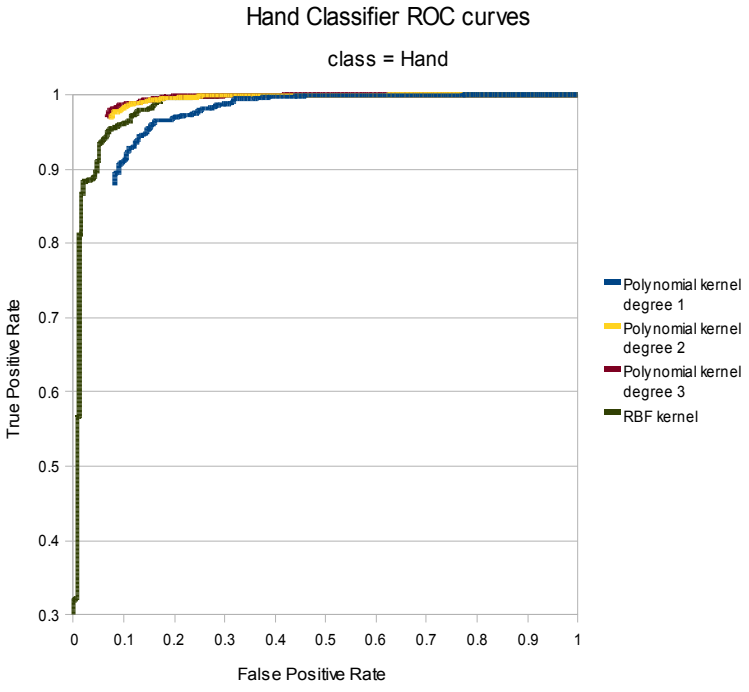

Figure 21. ROC curves for hand class in the hand classifier for different kernels in the SVM-SMO algorithm.

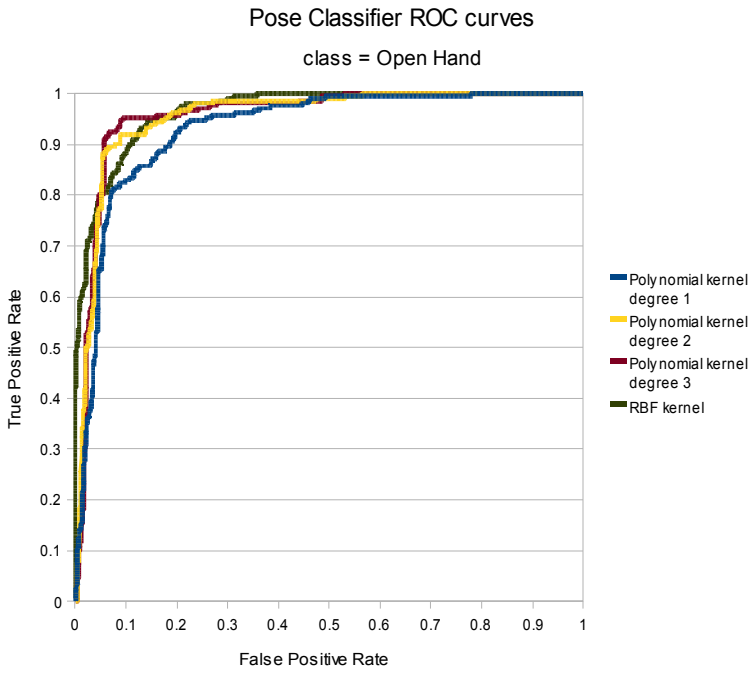

Figure 22. ROC curves for the open hand class in the pose classifier for different kernels in the SVM-SMO algorithm.

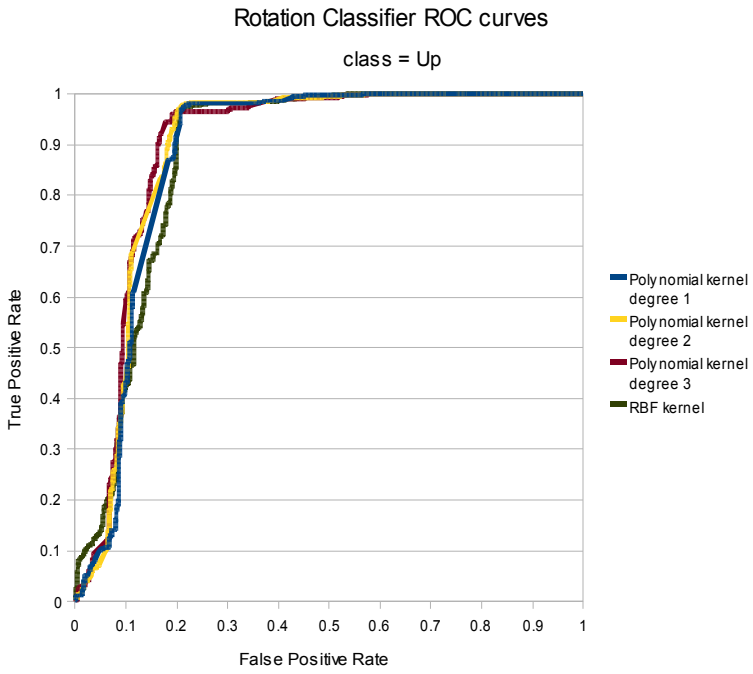

Figure 23. ROC curves for the open hand class in the pose classifier for different kernels in the SVM-SMO algorithm.

From this last set of experiments, it is clear that more complex kernel spaces cause more accurate classification results, meaning that the sample space is better divided when working with these more complex spaces. However, the cost of working with these more complex kernels, especially during the training phase, should be considered as the training time was much higher when using the complex kernels—sometimes more than four times as long. Table 6 shows the required training time for all classifiers and the time to test the just-trained classifier over the same testing data. This is done to validate the training process (classifiers are expected to accurately classify more than 98% of its own training data) and to obtain a reference of the required evaluation time of the classifiers.

Classifier	Training Time	Cross Validation Testing Time
	Seconds	Seconds
	Hand Classifier	
SVM-SMO polynomial kernel degree 1	1210.48	2.08
Naïve Bayes	0.05	0.41
J48 Tree	0.47	0.28
Adaboost with J48	4.85	0.22
Adaboost with ID3	15.85	0.55
SVM-SMO polynomial kernel degree 2	5041.53	822.36
SVM-SMO polynomial kernel degree 3	6538.80	1022.63
Svm-SMO RBF kernel	6359.23	1881.51
	Pose Classifier	
SVM-SMO polynomial kernel degree 1	6217.73	26.68
Naïve Bayes	0.14	0.89
J48 Tree	1.89	0.30
Adaboost with J48	16.32	0.48
Adaboost with ID3	35.12	0.92
SVM-SMO polynomial kernel degree 2	21873.49	14371.52
SVM-SMO polynomial kernel degree 3	30489.14	19816.76
Svm-SMO RBF kernel	7202.49	6656.90
	Rotation Classifier	
SVM-SMO polynomial kernel degree 1	113.24	4.43
Naïve Bayes	0.05	0.31
J48 Tree	0.28	0.17
Adaboost with J48	2.96	0.16
Adaboost with ID3	8.56	0.30
SVM-SMO polynomial kernel degree 2	622.86	647.23
SVM-SMO polynomial kernel degree 3	858.89	1001.60
Svm-SMO RBF kernel	171.32	240.09

Table 6. Training time and cross validation testing time for all classifiers considered in this work.

4. The Application: Implementing 3D gestures

The initial claim that justified focusing on 3D gestures of hand, pose, and rotation recognition was that if the latter information was known for each frame, then building and recognizing the 3D gestures is a simple task. To prove this and to build and recognize 3D gestures using the classifiers presented in the previous section, two applications that require 3D gestures were successfully implemented using the system described in this paper.

The first application called ZCam driver for the open source Sparsh UI [29] framework was developed as part of this same work with the previously stated purpose of proving that 3D gestures can be recognized using hand, pose, and rotation recognition. (This application will be described in detail in this section.)

The second application was developed by an independent team Kodavali S., Patel A., and Owusu E [32]. This application used the three classifiers system described in the previous sections and the training samples generated during the performance evaluation to develop a 3D modeling system. This system helps the user to model 3D objects by using the ZCam as the main interface and a series of defined gestures to interact with the model in the screen. Figure 24 shows the application in action where the user is using a pointing gesture to translate the 3D model. Another gesture was used for rotation. A demonstration video of the application can be seen on youtube.com [31].

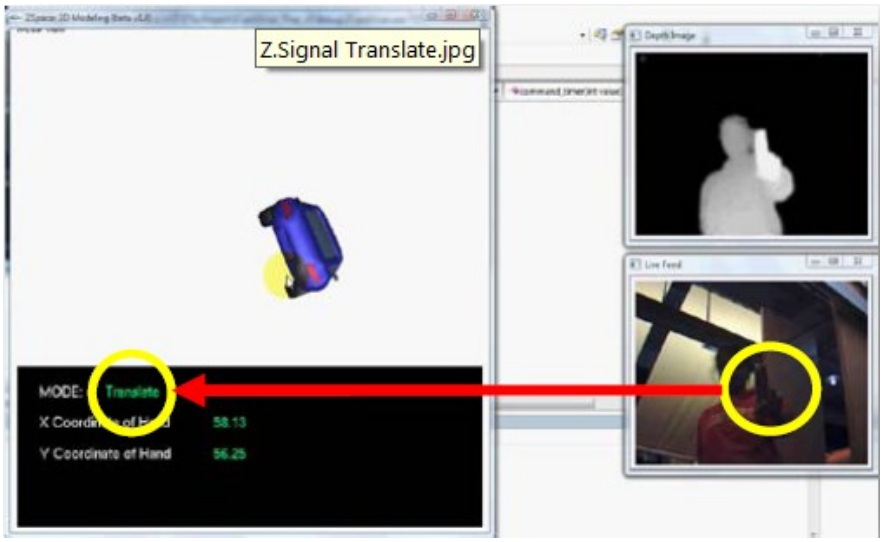

Figure 24. Kodavali S., Patel A., and Owusu E 3D modeling application.
Image obtained from [32]

The goal of the of the ZCam driver for the Sparsh UI application is to extend the Sparsh UI framework so that current and new Sparsh-UI client applications will be able to run with a ZCam as if it were a simulated multi-touch device. This integrates two important interface technologies: touch and computer vision.

Sparsh-UI is a platform independent multi-touch framework which allows client applications to run over different hardware in a transparent manner for the application. The Sparsh-UI solution consists of three main components: the input device, the gesture sever, and the gesture adapter.

The input device can be thought of as the hardware interface of the system. As part of the input device there are drivers already developed for several kinds of hardware, including an optical FTIR system, an infrared bezel, and others. The goal for this system is to add the ZCam to already supported devices, even though it is not a multi-touch device.

The gesture server is the main component of the system and is the one in charge of translating the input data provided by the input device into gestures. The gesture server has

several of the most common gestures used in multi-touch devices already implemented, such as drag, rotate, zoom, and more. Additionally, this component is designed in such a way that new gestures can be added easily.

Finally, once a gesture has been identified, the information is sent to the gesture adapter that formats the information so that the client applications can understand them. The gesture adapter is the client interface of the system.

Figure 25 shows a high level diagram of the Sparsh-UI data flow. Data, in the form of touch points, are input to the system through the input device (1). One touch point represents the event that goes from a user making contact to the multi-touch surface until the contact ends. This means that one touch point is actually composed of several points of contact which occur in time. Once a new touch point is detected, the gesture server will ask all registered clients to claim the point (2), meaning that the client would be prompted to either return an ID of the object being touched or to ignore this touch point. Client applications can subscribe their components to any arbitrary list of gestures—at this point, this information is also prompted by the gesture server. As long as the touch point evolves, new points are received for the touch point, (3) the gesture server would start recognizing gestures (4), if the component that claimed the touch point is registered for any of those gestures, gesture events would be sent to the client for that component through the gesture adapter (5).

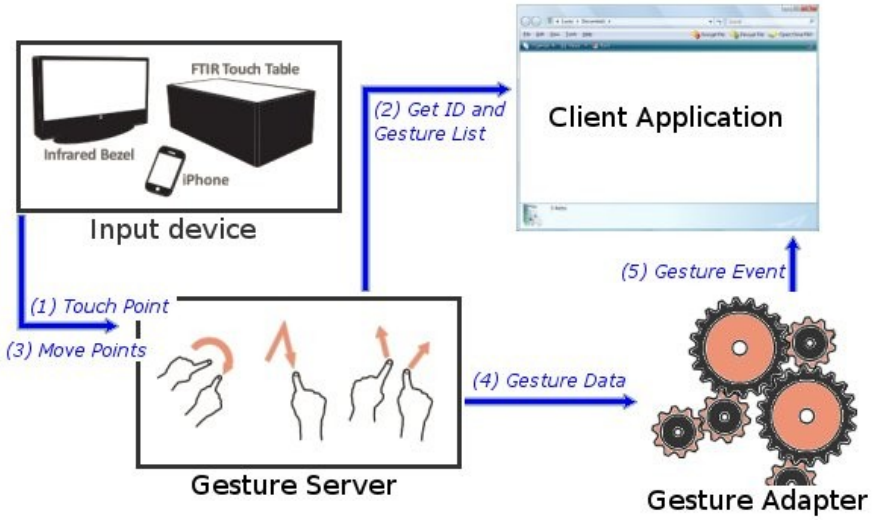

Figure 25. Schema of Sparsh-UI main functionalities.
Some images of this figure were obtained from Sparsh-UI's web site:
http://code.google.com/p/sparsh-ui

So, the goal of the application can be reformulated to include the ZCam as one of the supported input devices so that client applications can interact with it transparently. To achieve this goal, the information provided by the three classifiers (hand, pose, and rotation) plus the position in space of the closest point previously rescaled to [0,1] was used as input. All this information was processed through a finite state machine that would generate either a new type of event called a hover event, or would generate the required touch points as any other touch device would.

One of the main differences when using the ZCam versus using a touch device is that with touch devices, the user makes contact exactly in the point he or she wants, whereas with the camera, the user's hand is always present so the user must be able to move his or her hand freely without generating touch points as would happen when moving his or her hand over the touch device without making contact with it. To do so, a new event was added to the set of existing Sparsh gesture events called “hover event.” The new event informs the client

application that the user is moving his or her hand around the screen but is not actually touching anything. The client application can use this event and the information it provides to show the user where the hand is with respect to the application's screen.

The transition between states of the state machine can result from two possible causes. First, a transition can occur because a new hand pose or a new rotation angle or a combination of them has been identified. This transition will be complete as soon as the new state has been identified. Second, a transition can occur as a result of an internal processing of the state. Once a new state is reached, if the state was set with a processing function, then this function is run. As a consequence, processing a jump to another state may occur. Note that with the first transition type, the machine waits in the current state until new information arrives, whereas with the second transition type, jumps between states are done automatically without waiting for more information. This is so the machine does not stop on those states governed by this type of transition.

Figure 26 shows a schematic view of the implemented state machine (not all transitions are included for clarity purposes) that covers drag and rotation gestures. Other gestures can be added by just extending this state machine. The first type transitions are drawn in blue and the second type transitions are drawn in green.

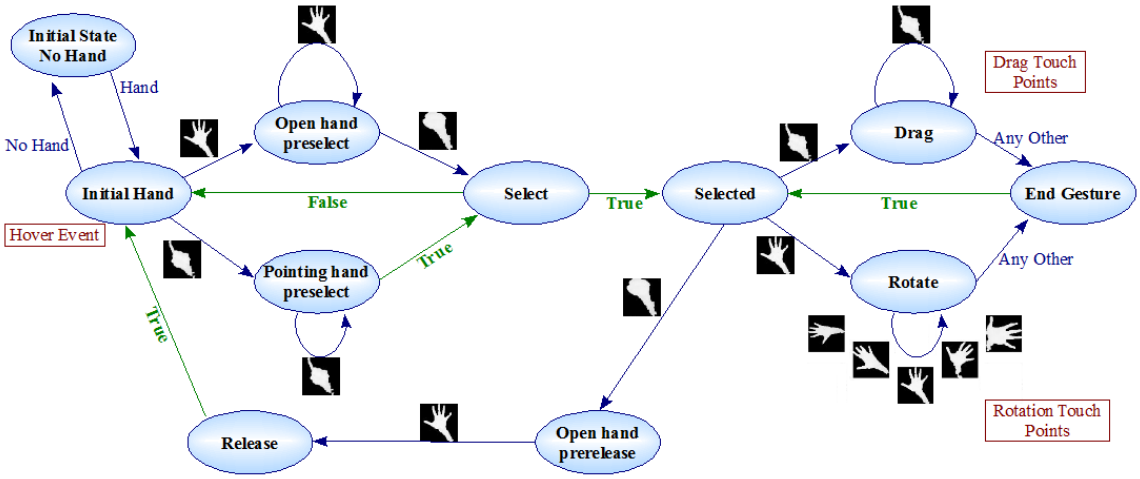

Figure 26. State machine for drag and rotate gestures of the ZCam driver for Sparsh. Other gestures can be added by extending this state machine.
(Not all transitions are included in this illustration of the state machine for clarity purposes.)

The state machine starts at the initial state of "No Hand." Once the user's hand is detected, the machine moves to the "Hand" state where the "hover event" is sent. The user can select a component of the application anytime. To do this, the user either points to the object and moves forward, or opens and closes his or her hand as if grabbing the component. Once a component is selected, the user can either drag the object around or rotate it. Either way, the relevant touch points are generated so the Sparsh-UI framework sends the corresponding events to the application. Notice that even in the event of a "No Hand," the object still keeps being selected, allowing the user to rest between operations. Finally, if the user decides to release the object, he or she can perform the relevant sequence.

Using the driver, both drag and rotate events were successfully sent to a previous existing application "Sparsh Tangrams," with just some minor changes in the application (support for the hover event). This indicates that application was successful and that the goal of proving that 3D gesture recognition can easily be done when hand, pose, and rotation information is available was achieved. The ZCam driver has successfully been integrated within the Sparsh

UI framework and will be included in future releases of the framework once all the supported Sparsh gestures have been mapped to 3D gestures in a similar fashion as drag and rotate. To do this, a new state will be created. It can be reached by transitioning from the Selected state through some hand pose and will transition to the End Gesture state once the gesture has ended. Although the application works correctly, a scaling problem must be solved so that the Spash UI client applications can be easily used with the ZCam. The camera image definition 320x240 is much smaller than the application 1280x768, so small hand movements result in big jumps in the application, making it difficult to perform precise movements. Also, it was suggested to adjust the scale dynamically so once an object is selected, the user can then perform big movements on the screen with small wrist movements. The idea is to reduce the arm movements during the gestures and avoid tiring the arm. Both problems would be addressed in future work done to the Sparsh UI framework. However, it is important to highlight that these scaling problems are not gesture recognition problems, but enhancements required to improve the relationship between Sparsh UI and ZCam. These problems do not compromise the fact that 3D gestures were successfully recognized and that the goal of this application was achieved.

5. Conclusions and future work

This research was framed by the following question: Is it possible to create a complete gesture recognition system using the ZCam which recognizes previously-trained hand gestures in real time and does not require any other interaction from the user? The question was answered by creating a system which uses the ZCam and three previously trained classifiers to recognize gestures in real time and does not require any other interaction from the user because it is able to detect the user's presence and when he or she begins to perform a gesture.

Using the approach of modeling 3D gestures as a sequence of hand poses, a complete solution for the one-hand 3D gesture recognition problem was proposed, implemented, and proved to be reliable. It is a complete solution because the 3D gesture recognition problem was not addressed as an isolated problem, but was considered in the context of a real application. In this sense, the common assumption that the system starts its life cycle with the user's hand in a starting position was not used; in fact, the assumption that the user's hand was present in the scene was not used either. So the proposed solution first addresses the problem of analyzing the scene and interpreting whether what it sees is actually the user's hand. If that is the case, then pose and rotation recognition is done in each frame. Finally, all the information is processed together using a finite state machine to generate the 3D gesture.

Considering the importance of applying 3D recognition in real time, it became clear that the required image processing must be as simple as possible. Therefore, a novel and simple feature set was proposed and evaluated. This can be obtained using a small number of image processing operations. Based on the accuracy results obtained during the performance

evaluation, the feature set seems to model the samples space in such a way that good level of classification can be achieved.

Additionally, as a result of the evaluations, it can be concluded that overall the SVM-SMO classification algorithm has the best performance. Moreover, if further accuracy is needed, more complex kernels can be used only requiring some extra effort during the training process which needs to be executed only once and off line.

Finally, the integration into the Sparsh UI framework created a connection between the 3D gesture recognition through Computer Vision and 2D gesture recognition done on touch surfaces. Actually, using the previously mentioned integration applications (which were originally were designed to run over touch surfaces) can now be used with much less expensive hardware.

A demonstration video of the recognition system where the three classifiers are shown independently can be seen on youtube.com [24].

5.1.1. Future Work

The goal of this research was to present a complete solution for the 3D gesture recognition problem and show it to be reliable. However, there is room for improvement and for new experiments. Some of these new opportunities have already been identified.

The system developed in this research was trained with only the right hand, so it only processes right-handed gesture. However, it would be interesting to include left-handed gestures, which could be done by either mirroring the image and then comparing the classification results of the original image and the mirrored, or by adding the necessary samples to the current training set.

It would also be interesting to develop an algorithm to dynamically define the best depth window size when thresholding from the closest point. In this research, a depth window of 20 levels of gray was used; however, being able to dynamically adjust the window's size would help to obtain more accurate classification. This problem has several challenges. First, the required depth depends not only upon the hand's distance from the camera, but also on the hand's pose; whereas a hand pointing towards the camera would require a large depth window, a hand showing its palm to the camera can be solved with a much smaller window. Possibly considering previous frames and their hand pose classification would help develop a window that would better fit the current hand position. Another option is to use a similar approach, such as Malassiotis and Strintzis [30], and apply some variation of the threshold, cluster, and merge algorithm.

Frames can also be used to improve the performance of the classifiers in the current frame. One approach for this could be the application of Hidden Markov Models after having the initial classification to either validate or change the initial classification. To apply this idea, they should return the probability distribution over the different classes and allow the HMM machine to finally decide the classification class instead of having the current classifiers returning the resulting class.

Another future improvement would be to analyze the impact of having more granularity when building the feature set. Currently, only 65 features are build-based in the 8x8 windows in which the input image is divided. It would be interesting to analyze the impact of having 256 features by building 4x4 windows, while leaving the rest of the functionalities as they currently work.

Finally, the natural next step is to generate more test cases with different users to cross

validate the results obtained in this work. Additionally, a usability test with several trained and untrained users would give valuable information about the affordances of the 3D gestures and the usability of the ZCam driver for Sparsh UI applications.

6. References

1. Al-Rajab, Moaath and Hogg, David and Ng, Kia. A Comparative Study on Using Zernike Velocity Moments and Hidden Markov Models for Hand Gesture Recognition. *Articulated Motion and Deformable Objects*, 5098. 319-327, **2008**

2. Alon, Jonathan and Athitsos, Vassilis and Yuan, Quan and Sclaroff, Stan. Simultaneous Localization and Recognition of Dynamic Hand Gestures. *Motion and Video Computing, 2005. WACV/MOTIONS '05 Volume 2. IEEE Workshop on*, 2. 254-260, **2005**

3. Athitsos, V. and Sclaroff, S.. Estimating 3D hand pose from a cluttered image. , 2. II-432-9 vol.2, **2003**

4. Bernhard Scholkopf and Patrice Simardz and Alex Smolay and Vladimir Vapnikz. Prior Knowledge in Support Vector Kernels. *Advances in neural information processing systems*, . , **1998**

5. Björn Stenger. Template-Based Hand Pose Recognition Using Multiple Cues. *Springer Berlin / Heidelberg*. . **2006**

6. Caifeng Shan and Tieniu Tan and Yucheng Wei. Real-time hand tracking using a mean shift embedded particle filter. *Pattern Recognition*, 40. 1958 - 1970, **2007**

7. Caifeng Shan and Yucheng Wei and Tieniu Tan and Ojardias, F.. Real time hand tracking by combining particle filtering and mean shift. , . 669-674, **2004**

8. Christopher M. Bishop. Pattern Recognition and Machine Learning. *Springer*. M. Jordan and J. Kleinberg and B. Schölkopf. **2006**

9. Dan Luo and Jun Ohya. Hand-gesture extraction and recognition from the video sequence acquired by a dynamic camera using condensation algorithm. *Intelligent Robots and*

Computer Vision XXVI: Algorithms and Techniques, 7252. 72520S, **2009**

10. E. Osuna and R. Freund and F. Girosi. Improved training algorithm for support vector machines. *Proc. IEEE NNSP*, . , **1997**

11. Feng-Sheng Chen and Chih-Ming Fu and Chung-Lin Huang. Hand gesture recognition using a real-time tracking method and hidden Markov models. *Image and Vision Computing*, 21. 745 - 758, **2003**

12. G. J. Iddan \& G. Yahav. 3D Imaging in the studio. *The Society of Photo-Optical Instrumentation Engineers*, 4298. 48, **2000**

13. Hideto Oda and Bilan Zhu and Junko Tokuno and Motoki Onuma and Akihito Kitadai and Masaki Nakagawa. A Compact On-line and Off-line Combined Recognizer. *Tenth International Workshop on Frontiers in Handwriting Recogntion*, 1. 133–138, **2006**

14. Holsti, OR. Content analysis for the social sciences and humanities. *Addison-Wesley Pub. Co*. . **1969**

15. Huang, Jin and Ling, Charles X.. Using AUC and Accuracy in Evaluating Learning Algorithms. *IEEE Trans. on Knowl. and Data Eng.*, 17. 299--310, **2005**

16. Ian H. Witten and Eibe Frank. Data Mining: Practical machine learning tools and techniques. . Morgan Kaufmann. **2005**

17. J. Holub and B. Nekolny and M. Van Waardhuizen. Recognition of American Sign Language using the 3DV ZCam. *http://www.vrac.iastate.edu/575x/S09/doku.php?id=projects:project1*, . , **2009**

18. Jianjun Ye and Hongxun Yao and Feng Jiang. Based on HMM and SVM multilayer architecture classifier for Chinese sign language recognition with large vocabulary. , . 377-380, **2004**

19. JR Quinlan. C4.5: Programs for Machine Learning. *Publishers Inc., San Francisco, CA, USA*. Morgan Kaufmann. **1993**

20. Kaustubh Srikrishna Patwardhan and Sumantra Dutta Roy. Hand gesture modelling and recognition involving changing shapes and trajectories, using a Predictive EigenTracker. *Pattern Recognition Letters*, 28. 329 - 334, **2007**

21. Kikuo Fujimura and Lijie Xu. Sign Recognition Using Constrained Optimization. *Springer Berlin / Heidelberg*. . **2007**

22. Lee, Jaeseon and Park, Kyoung and Hahn, Minsoo. The 3D Sensor Table for Bare Hand Tracking and Posture Recognition. *Advances in Multimedia Modeling*, 4351. 138--146, **2006**

23. Lizhong Gu and Jianbo Su. Natural hand posture recognition based on Zernike moments and hierarchical classifier. *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, . 3088-3093, **2008**

24. Lucas Bonansea. Demonstration video of the 3D gesture recognition system using Zcam and SVM. http://www.youtube.com/watch?v=VsM0a_3I1_Q. **2009**

25. Marcel, S. and Bernier, O. and Viallet, J.-E. and Collobert, D.. Hand gesture recognition using input-output hidden Markov models . , . 456-461, **2000**

26. Nitesh V. Chawla and Kevin W. Bowyer and Lawrence O. Hall and W. Philip Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16. 321-357, **2002**

27. Platt, John C.. Fast training of support vector machines using sequential minimal optimization. , . 185--208, **1999**

28. Rafael Jordan-Osorio and Vlad Sukhoy. Proteins visualization control using hand gestures. http://www.vrac.iastate.edu/575x/S09/doku.php?id=projects:project2. **2009**

29. Ramanahally, P. and Gilbert, S. and Anagnost, C. and Niedzielski, T. and Velázquez, D. Creating a Collaborative Multitouch Computer Aided Design Program. *Proc. of WinVR*, . , **2009**

30. S. Malassiotis and M.G. Strintzis. Real-time hand posture recognition using range data. *Image and Vision Computing*, 26. 1027 - 1037, **2008**

31. Sateesh Kodavali and Ankit Patel and Emmanuel Owusu. Demonstration video of the application "Z-Space: 3D Modeling Using Hand Gestures".. http://www.youtube.com/watch?v=yG_gt86Sghg. **2009**

32. Sateesh Kodavali and Ankit Patel and Emmanuel Owusu. Z-Space: 3D Modeling Using Hand Gestures. http://www.vrac.iastate.edu/575x/S09/doku.php?id=projects:project10. **2009**

33. Segen, J. and Kumar, S.. Shadow gestures: 3D hand pose estimation using a single camera. , 1. -485 Vol. 1, **1999**

34. Sushmita Mitra and Tinku Acharya. Gesture Recognition: A Survey. *Systems, Man and Cybernetics - Part C: Applications and Reviews. IEEE Transactions on*, 37. 311-324, **2007**

35. Thad Starner and Alex Pentland. Real-Time American Sign Language Recognition from Video Using Hidden Markov Models. *AAAI Technical Report FS-96-05*, . , **1996**

36. Thad Starner and Joshua Weaver and Alex Pentland. Real-Time American Sign Language Recognition Using Desk and Wearable Computer Based Video. *IEEE PAMI '98*, . , **1998**

37. Yen-Ting Chen and Kuo-Tsung Tseng. Multiple-angle Hand Gesture Recognition by Fusing SVM Classifiers. , . 527-530, **2007**

38. Yikai Fang and Kongqiao Wang and Jian Cheng and Hanqing Lu. A Real-Time Hand Gesture Recognition Method. *Multimedia and Expo, 2007 IEEE International Conference on*, . 995-998, **2007**

39. Ying Wu and Huang, T.S.. Capturing articulated human hand motion: a divide-and-conquer approach. , 1. 606-611 vol.1, **1999**

40. Ying Wu and Huang, T.S.. View-independent recognition of hand postures. , 2. 88-94 vol.2, **2000**

41. Youbin Chen and Xiaoqing Ding and Youshou Wu. Off-line handwritten Chinese character recognition based on crossing line feature. , 1. 206-210 vol.1, **1997**

42. Yun Liu and Zhijie Gan and Yu Sun. Static Hand Gesture Recognition and its Application based on Support Vector Machines. , . 517-521, **2008**

43. Zhenyao Mo and Ulrich Neumann. Real-time Hand Pose Recognition Using Low-Resolution Depth Images. *Int. Conf. on Computer Vision and Pattern Recognition*, . , **2006**